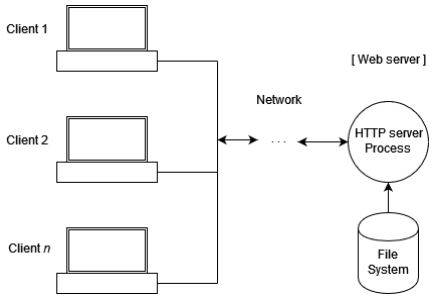
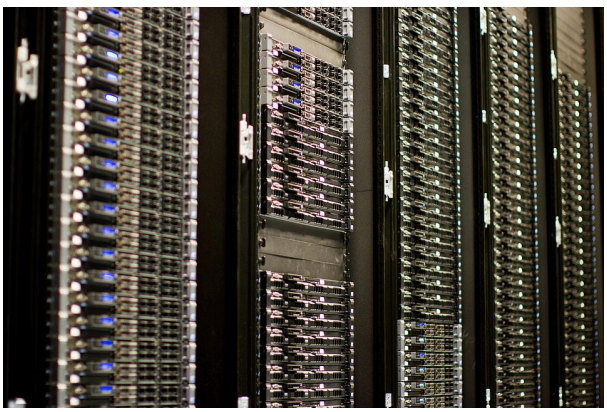
Web server

A **web server** is computer software and underlying hardware that accepts requests via HTTP (the network protocol created to distribute web content) or its secure variant HTTPS. A user agent, commonly a web browser or web crawler, initiates communication by making a request for a web page or other resource using HTTP, and the server responds with the content of that resource or an error message. A web server can also accept and store resources sent from the user agent if configured to do so.[1][2][3][4][5]

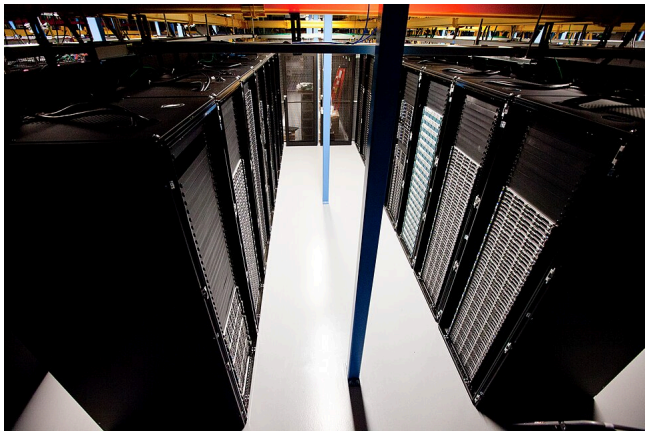


PC clients communicating via the network with a web server serving static content only

The inside and front of a Dell PowerEdge server, a computer designed to be mounted in a rack mount environment. Servers similar to this one are often used as web servers.



Multiple web servers may be used for a high-traffic website.



Server farm with thousands of web servers used for super-high traffic websites

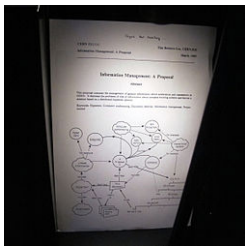
ADSL modem running an embedded web server serving dynamic web pages used for modem configuration

The hardware used to run a web server can vary according to the volume of requests that it needs to handle. At the low end of the range are embedded systems, such as a router that runs a small web server as its configuration interface. A high-traffic Internet website might handle requests with hundreds of servers that run on racks of high-speed computers.[6]

A resource sent from a web server can be a pre-existing file (static content) available to the web server, or it can be generated at the time of the request (dynamic content) by another program that communicates with the server software. The former usually can be served faster and can be more easily cached for repeated requests, while the latter supports a broader range of applications.

Technologies such as REST and SOAP, which use HTTP as a basis for general computer-to-computer communication, as well as support for WebDAV extensions, have extended the application of web servers well beyond their original purpose of serving human-readable pages.

History



First web proposal (1989) evaluated as *"vague but exciting..."*

The world's first web server, a NeXT Computer workstation with Ethernet, 1990. The case label reads: "This machine is a server. DO NOT POWER IT DOWN!!"

This is a very brief history of *web server programs*, so some information necessarily overlaps with the histories of the *web browsers*, the *World Wide Web* and the *Internet*; therefore, for the sake of clarity and understandability, some key historical information below reported may be similar to that found also in one or more of the above-mentioned history articles.[7]

Initial WWW project (1989–1991)

In March 1989, Sir Tim Berners-Lee proposed a new project to his employer CERN, with the goal of easing the exchange of information between scientists by using a hypertext system. The proposal titled *"HyperText and CERN"*, asked for comments and it was read by several people. In October 1990 the proposal was reformulated and enriched (having as co-author Robert Cailliau), and finally, it was approved.[8][9][10]

Between late 1990 and early 1991 the project resulted in Berners-Lee and his developers writing and testing several software libraries along with three programs, which initially ran on NeXTSTEP OS installed on NeXT workstations:[11][12][10]

- a graphical web browser, called WorldWideWeb;
- a portable line mode web browser;

- a web server, later known as CERN httpd.

Those early browsers retrieved web pages written in a simple early form of HTML from web servers using a new basic communication protocol that was named HTTP 0.9.

In August 1991 Tim Berners-Lee announced the *birth of WWW technology* and encouraged scientists to adopt and develop it.[13] Soon after, those programs, along with their source code, were made available to people interested in their usage.[11] Although the source code was not formally licensed or placed in the public domain, CERN informally allowed users and developers to experiment and further develop on top of them. Berners-Lee started promoting the adoption and the usage of those programs along with their porting to other operating systems.[10]

Fast and wild development (1991–1995)

In December 1991, the **first web server outside Europe** was installed at SLAC (U.S.A.).[12] This was a very important event because it started trans-continental web communications between web browsers and web servers.

In 1991–1993, CERN web server program continued to be actively developed by the www group, meanwhile, thanks to the availability of its source code and the public specifications of the HTTP protocol, many other implementations of web servers started to be developed.

In April 1993, CERN issued a public official statement stating that the three components of Web software (the basic line-mode client, the web server and the library of common code), along with their source code, were put in the public domain.[14] This statement freed web server developers from any possible legal issue about the development of *derivative work* based on that source code (a threat that in practice never existed).

At the beginning of 1994, the most notable among new web servers was NCSA httpd which ran on a variety of Unix-based OSs and could serve *dynamically generated content* by implementing the `POST` HTTP method and the CGI to communicate with external programs. These capabilities, along with the multimedia features of NCSA's Mosaic browser (also able to manage HTML FORMs in order to send data to a web server) highlighted the potential of web technology for publishing and distributed computing applications.

Number of active web sites (1991–1996)[15][16]

In the second half of 1994, the development of NCSA httpd stalled to the point that a group of external software developers, webmasters and other professional figures interested in that server, started to write and collect patches thanks to the NCSA httpd source code being available to the public domain. At the beginning of 1995 those patches were all applied to the last release of NCSA source code and, after several tests, the Apache HTTP server project was started.[17][18]

At the end of 1994, a new commercial web server, named Netsite, was released with specific features. It was the first one of many other similar products that were developed first by Netscape, then also by Sun Microsystems, and finally by Oracle Corporation.

In mid-1995, the first version of IIS was released, for Windows NT OS, by Microsoft. This marked the entry, in the field of World Wide Web technologies, of a very important commercial developer and vendor that has played and still is playing a key role on both sides (client and server) of the web.

In the second half of 1995, CERN and NCSA web servers started to decline (in global percentage usage) because of the widespread adoption of new web servers which had a much faster development cycle along with more features, more fixes applied, and more performances than the previous ones.

Explosive growth and competition (1996–2014)

Number of active web sites (1996-2002)[16][19]

Sun's Cobalt Qube 3 – a computer server appliance (2002, discontinued)

At the end of 1996, there were already over fifty known, different web-server-software programs that were available to everybody who wanted to own an Internet domain name or to host websites.[20] Many of them lived only shortly and were replaced by other web servers.

The publication of RFCs about protocol versions HTTP/1.0 (1996) and HTTP/1.1 (1997, 1999), forced most web servers to comply (not always completely) with those standards. The use of TCP/IP persistent connections (HTTP/1.1) required web servers both to increase the maximum number of concurrent connections allowed and to improve their level of scalability.

Between 1996 and 1999, Netscape Enterprise Server and Microsoft's IIS emerged among the leading commercial options whereas among the freely available and open-source programs Apache HTTP Server held the lead as the preferred server (because of its reliability and its many features).

In those years there was also another commercial, highly innovative and thus notable web server called Zeus (*now discontinued*) that was known as one of the fastest and most scalable web servers available on market, at least till the first decade of 2000s, despite its low percentage of usage.

Apache resulted in the most used web server from mid-1996 to the end of 2015 when, after a few years of decline, it was surpassed initially by IIS and then by Nginx. Afterward IIS dropped to much lower percentages of usage than Apache (see also market share).

From 2005–2006, Apache started to improve its speed and its scalability level by introducing new performance features (e.g., event MPM and new content cache).[21][22] As those new performance improvements initially were marked as experimental, they were not enabled by its users for a long time and so Apache suffered, even more, the competition of commercial servers and, above all, of other open-source servers which meanwhile had already achieved far superior performances (mostly when serving static content) since the beginning of their development and at the time of the Apache decline were able to offer also a long enough list of well tested advanced features.

A few years after 2000 started, not only other commercial and highly competitive web servers (e.g., LiteSpeed) but also many other open-source programs such as Hiawatha, Cherokee HTTP server, Lighttpd, Nginx and other derived and related products also available with commercial support emerged.

Around 2007–2008, most popular web browsers increased their previous default limit of 2 persistent connections per host-domain (a limit recommended by RFC-2616)[23] to 4, 6 or 8 persistent connections per host-domain, in order to speed up the retrieval of heavy web pages with lots of images, and to mitigate the problem of the shortage of persistent connections dedicated to dynamic objects used for bi-directional notifications of events in web pages.[24] Within a year, these changes, on average, nearly tripled the maximum number of persistent connections that web servers had to manage. This trend (of increasing the number of persistent connections) definitely gave a strong impetus to the adoption of reverse proxies in front of slower web servers and it gave also one more chance to the emerging new web servers that could show all their speed and their capability to handle very high numbers of concurrent connections without requiring too many hardware resources (expensive computers with lots of CPUs, RAM and fast disks).[25]

New challenges (2015 and later years)

In 2015, RFCs published new protocol version [HTTP/2], and as the implementation of new specifications was not trivial at all, a **dilemma arose among developers of less popular web**

servers (e.g., with a percentage of usage lower than 1% .. 2%), about adding or not adding support for that new protocol version.[26][27]

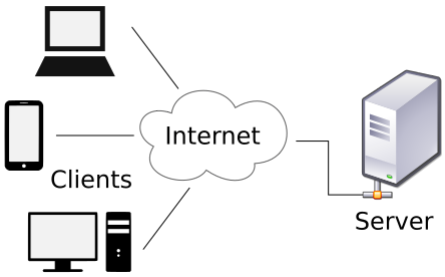
In fact supporting HTTP/2 often required radical changes to their internal implementation due to many factors (practically always required encrypted connections, capability to distinguish between HTTP/1.x and HTTP/2 connections on the same TCP port, binary representation of HTTP messages, message priority, compression of HTTP headers, use of streams also known as TCP/IP sub-connections and related flow-control, etc.) and so a few developers of those web servers opted for not supporting new HTTP/2 version (at least in the near future) also because of these main reasons:[26][27]

- protocols HTTP/1.x would have been supported anyway by browsers for a very long time (maybe forever) so that there would be no incompatibility between clients and servers in next future;
- implementing HTTP/2 was considered a task of overwhelming complexity that could open the door to a whole new class of bugs that till 2015 did not exist and so it would have required notable investments in developing and testing the implementation of the new protocol;
- adding HTTP/2 support could always be done in future in case the efforts would be justified.

Instead, developers of **most popular web servers, rushed to offer the availability of new protocol**, not only because they had the work force and the time to do so, but also because usually their previous implementation of SPDY protocol could be reused as a starting point and because most used web browsers implemented it very quickly for the same reason. Another reason that prompted those developers to act quickly was that webmasters felt the pressure of the ever increasing web traffic and they really wanted to install and to try – as soon as possible – something that could drastically lower the number of TCP/IP connections and speedup accesses to hosted websites.[28]

In 2020–2021 the HTTP/2 dynamics about its implementation (by top web servers and popular web browsers) were partly replicated after the publication of advanced drafts of future RFC about HTTP/3 protocol.

Technical overview



PC clients connected to a web server via Internet

The following technical overview should be considered only as an attempt to give a few very *limited examples* about *some* features that may be implemented in a web server and *some* of the tasks that it may perform in order to have a sufficiently wide scenario about the topic.

A **web server program** plays the role of a server in a client–server model by implementing one or more versions of HTTP protocol, often including the HTTPS secure variant and other features and extensions that are considered useful for its planned usage.

The complexity and the efficiency of a web server program may vary a lot depending on:[1]

- common features implemented;
- common tasks performed;
- performances and scalability level aimed as a goal;
- software model and techniques adopted to achieve wished performance and scalability level;
- target hardware and category of usage (e.g., embedded system, low-medium traffic web server, high traffic Internet web server).

Common features

Although web server programs differ in how they are implemented, most of them offer the following common features.

These are **basic features** that most web servers usually have.

- Static content serving: to be able to serve static content (web files) to clients via HTTP protocol.
- HTTP: support for one or more versions of HTTP protocol in order to send versions of HTTP responses compatible with versions of client HTTP requests, (e.g., HTTP/1.0, HTTP/1.1, HTTP/2, HTTP/3).

- Logging: usually web servers have also the capability of logging some information, about client requests and server responses, to log files for security and statistical purposes.

A few other more **advanced** and popular **features** (*only a very short selection*) are the following ones.

- Dynamic content serving: to be able to serve dynamic content (generated on the fly) to clients via HTTP protocol.
- Virtual hosting: to be able to serve many websites (domain names) using only one IP address.
- Authorization: to be able to allow, to forbid or to authorize access to portions of website paths (web resources).
- Content cache: to be able to cache content in order to speed up server responses;
- Large file support: to be able to serve files whose size is greater than 2 GB on 32 bit OS.
- Bandwidth throttling: to limit the speed of content responses in order to not saturate the network and to be able to serve more clients;
- Rewrite engine: to map parts of clean URLs (found in client requests) to their real names.
- Custom error pages: support for customized HTTP error messages.

Common tasks

A web server program, when it is running, usually performs several general **tasks**:[1]

- starts, optionally reads and applies settings found in its configuration files or elsewhere, optionally opens log file, starts listening to client connections and requests;
- optionally tries to adapt its general behavior according to its settings and its current operating conditions;
- manages **client connections** (accepting new ones or closing the existing ones as required);
- **receives** client requests (by reading HTTP messages):
  - reads and verify each HTTP request message;
  - usually performs URL normalization;
  - usually performs URL mapping (which may default to URL path translation);
  - usually performs URL path translation along with various security checks;
- **executes** or refuses requested HTTP method:
  - optionally manages URL authorizations;
  - optionally manages URL redirections;
  - optionally manages requests for **static resources** (file contents):

        - optionally manages directory index files;
        - optionally manages regular files;
    - optionally manages requests for **dynamic resources**:
        - optionally manages directory listings;
        - optionally manages program or module processing, checking the availability, the start and eventually the stop of the execution of external programs used to generate dynamic content;
        - optionally manages the communications with external programs or internal modules used to generate dynamic content;
- **replies** to client requests sending proper HTTP responses (e.g., requested resources or error messages) eventually verifying or adding HTTP headers to those sent by dynamic programs and modules;
- optionally **logs** (partially or totally) **client requests or its responses** to an external user log file or to a system log file by syslog, usually using common log format;
- optionally **logs process messages** about **detected anomalies or other notable events** (e.g., in client requests or in its internal functioning) using syslog or some other system facilities; these log messages usually have a debug, warning, error, alert level which can be filtered (not logged) depending on some settings, see also severity level;
- optionally generates **statistics** about web traffic managed or its performances;
- other custom tasks.

Read request message

Web server programs are able:[29] [30] [31]

- to read an HTTP request message;
- to interpret it;
- to verify its syntax;
- to identify known HTTP headers and to extract their values from them.

Once an HTTP request message has been decoded and verified, its values can be used to determine whether that request can be satisfied or not. This requires many other steps, including **security checks**.

URL normalization

Web server programs usually perform some type of URL normalization (URL found in most HTTP request messages) in order to:

- make resource path always a clean uniform path from root directory of website;
- lower security risks (e.g., by intercepting more easily attempts to access static resources outside the root directory of the website or to access to portions of path below website root directory that are forbidden or which require authorization);
- make path of web resources more recognizable by human beings and web log analysis programs (also known as log analyzers or statistical applications).

The term *URL normalization* refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed, including the conversion of the scheme and host to lowercase. Among the most important normalizations are the removal of "." and ".." path segments and adding trailing slashes to a non-empty path component.

URL mapping

> "URL mapping is the process by which a URL is analyzed to figure out what resource it is referring to, so that that resource can be returned to the requesting client. This process is performed with every request that is made to a web server, with some of the requests being served with a file, such as an HTML document, or a gif image, others with the results of running a CGI program, and others by some other process, such as a built-in module handler, a PHP document, or a Java servlet."[32]

In practice, web server programs that implement advanced features, beyond the simple *static content serving* (e.g., URL rewrite engine, dynamic content serving), usually have to figure out how that URL has to be handled as a:

- URL redirection, a redirection to another URL;
- *static request* of file content;
- *dynamic request* of:
  - directory listing of files or other sub-directories contained in that directory;
  - other types of dynamic request in order to identify the program or module processor able to handle that kind of URL path and to pass to it other URL parts, (i.e., usually path-info and query string variables).

One or more configuration files of web server may specify the mapping of parts of **URL path** (e.g., initial parts of file path, filename extension and other path components) to a specific URL handler (file, directory, external program or internal module).[33]

When a web server implements one or more of the above-mentioned advanced features then the path part of a valid URL may not always match an existing file system path under website directory tree (a file or a directory in file system) because it can refer to a virtual name of an internal or external module processor for dynamic requests.

URL path translation to file system

Web server programs are able to translate an URL path (all or part of it), that refers to a physical file system path, to an absolute path under the target website's root directory.[33]

Website's root directory may be specified by a configuration file or by some internal rule of the web server by using the name of the website which is the host part of the URL found in HTTP client request.[33]

Path translation to file system is done for the following types of web resources:

- a local, usually non-executable, file (static request for file content);
- a local directory (dynamic request: directory listing generated on the fly);
- a program name (dynamic requests that is executed using CGI or SCGI interface and whose output is read by web server and resent to client who made the HTTP request).

The web server appends the path found in requested URL (HTTP request message) and appends it to the path of the (Host) website root directory. On an Apache server, this is commonly `/home/www/website` (on Unix machines, usually it is: `/var/www/website`). See the following examples of how it may result.

URL path translation for a static file request

Example of a *static request* of an existing file specified by the following URL:

```
http://www.example.com/path/file.html
```

The client's user agent connects to `www.example.com` and then sends the following HTTP/1.1 request:

```
GET /path/file.html HTTP/1.1
Host: www.example.com
Connection: keep-alive
```

The result is the local file system resource:

```
/home/www/www.example.com/path/file.html
```

The web server then reads the file, if it exists, and sends a response to the client's web browser. The response will describe the content of the file and contain the file itself or an error message will return saying that the file does not exist or its access is forbidden.

URL path translation for a directory request (without a static index file)

Example of an implicit *dynamic request* of an existing directory specified by the following URL:

```
http://www.example.com/directory1/directory2/
```

The client's user agent connects to `www.example.com` and then sends the following HTTP/1.1 request:

```
GET /directory1/directory2 HTTP/1.1
Host: www.example.com
Connection: keep-alive
```

The result is the local directory path:

```
/home/www/www.example.com/directory1/directory2/
```

The web server then verifies the existence of the directory and if it exists and it can be accessed then tries to find out an index file (which in this case does not exist) and so it passes the request to an internal module or a program dedicated to directory listings and finally reads data output and sends a response to the client's web browser. The response will describe the content of the directory (list of contained subdirectories and files) or an error message will return saying that the directory does not exist or its access is forbidden.

URL path translation for a dynamic program request

For a *dynamic request* the URL path specified by the client should refer to an existing external program (usually an executable file with a CGI) used by the web server to generate dynamic content.[34]

Example of a *dynamic request* using a program file to generate output:

```
http://www.example.com/cgi-bin/forum.php?
action=view&orderby=thread&date=2021-10-15
```

The client's user agent connects to `www.example.com` and then sends the following HTTP/1.1 request:

```
GET /cgi-bin/forum.php?action=view&ordeby=thread&date=2021-10-15
HTTP/1.1
Host: www.example.com
Connection: keep-alive
```

The result is the local file path of the program (in this example, a PHP program):

```
/home/www/www.example.com/cgi-bin/forum.php
```

The web server executes that program, passing in the path-info and the query string `action=view&orderby=thread&date=2021-10-15` so that the program has the info it needs to run. (In this case, it will return an HTML document containing a view of forum entries ordered by thread from October 15, 2021). In addition to this, the web server reads data sent from the external program and resends that data to the client that made the request.

Manage request message

Once a request has been read, interpreted, and verified, it has to be managed depending on its method, its URL, and its parameters, which may include values of HTTP headers.

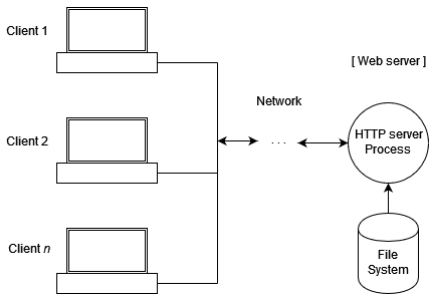
In practice, the web server has to handle the request by using one of these response paths:[33]

- if something in request was not acceptable (in status line or message headers), web server already sent an error response;
- if request has a method (e.g., `OPTIONS`) that can be satisfied by general code of web server then a successful response is sent;
- if URL requires authorization then an authorization error message is sent;
- if URL maps to a redirection then a redirect message is sent;
- if URL maps to a dynamic resource (a virtual path or a directory listing) then its handler (an internal module or an external program) is called and request parameters (query string and

path info) are passed to it in order to allow it to reply to that request;

- if URL maps to a static resource (usually a file on file system) then the internal static handler is called to send that file;
- if request method is not known or if there is some other unacceptable condition (e.g., resource not found, internal server error, etc.) then an error response is sent.

Serve static content



PC clients communicating via network with a web server serving static content only

If a web server program is capable of **serving static content** and it has been configured to do so, then it is able to send file content whenever a request message has a valid URL path matching (after URL mapping, URL translation and URL redirection) that of an existing file under the root directory of a website and file has attributes which match those required by internal rules of web server program.[33]

That kind of content is called *static* because usually it is not changed by the web server when it is sent to clients and because it remains the same until it is modified (file modification) by some program.

NOTE: when serving **static content only**, a web server program usually **does not change file contents** of served websites (as they are only read and never written) and so it suffices to support only these HTTP methods:

- `OPTIONS`
- `HEAD`
- `GET`

Response of static file content can be sped up by a **file cache**.

Directory index files

If a web server program receives a client request message with an URL whose path matches one of an existing *directory* and that directory is accessible and serving directory index files is enabled then a web server program may try to serve the first of known (or configured) static

index file names (a regular file) found in that directory; if no index file is found or other conditions are not met then an error message is returned.

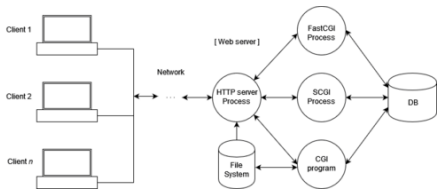
Most used names for static index files are: `index.html`, `index.htm` and `Default.htm`.

Regular files

If a web server program receives a client request message with an URL whose path matches the file name of an existing *file* and that file is accessible by web server program and its attributes match internal rules of web server program, then web server program can send that file to client.

Usually, for security reasons, most web server programs are pre-configured to serve only regular files or to avoid to use *special file types* like device files, along with symbolic links or hard links to them. The aim is to avoid undesirable side effects when serving static web resources.[35]

Serve dynamic content



PC clients communicating via network with a web server serving static and dynamic content

If a web server program is capable of **serving dynamic content** and it has been configured to do so, then it is able to communicate with the proper internal module or external program (associated with the requested URL path) in order to pass to it the parameters of the client request. After that, the web server program reads from it its data response (that it has generated, often on the fly) and then it resends it to the client program who made the request.

NOTE: when serving **static and dynamic content**, a web server program usually has to support also the following HTTP method in order to be able to safely **receive data** from clients and so to be able to host also websites with interactive forms that may send large data sets (e.g., lots of data entry or file uploads) to web server, external programs or modules:

- `POST`

In order to be able to communicate with its internal modules or external programs, a web server program must have implemented one or more of the many available **gateway interfaces** (see also Web Server Gateway Interfaces used for dynamic content).

The three **standard** and historical **gateway interfaces** are the following ones.

CGI

An external CGI program is run by web server program for each dynamic request, then web server program reads from it the generated data response and then resends it to client.

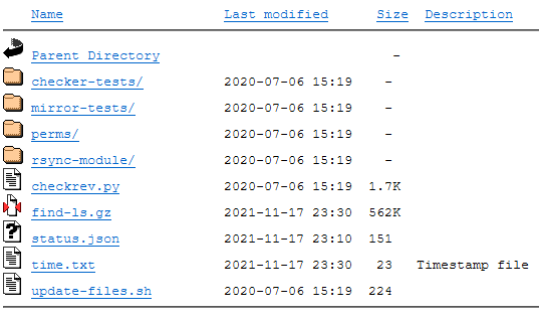
SCGI

An external SCGI program (it usually is a process) is started once by web server program or by some other program or process and then it waits for network connections; every time there is a new request for it, web server program makes a new network connection to it in order to send request parameters and to read its data response, then network connection is closed.

FastCGI

An external FastCGI program (it usually is a process) is started once by web server program or by some other program or process and then it waits for a network connection which is established permanently by web server; through that connection are sent the request parameters and read data responses.

Directory listings



Name	Last modified	Size	Description
Parent Directory		-	
checker-tests/	2020-07-06 15:19	-	
mirror-tests/	2020-07-06 15:19	-	
perms/	2020-07-06 15:19	-	
rsync-module/	2020-07-06 15:19	-	
checkrev.py	2020-07-06 15:19	1.7K	
find-ls.gz	2021-11-17 23:30	562K	
status.json	2021-11-17 23:10	151	
time.txt	2021-11-17 23:30	23	Timestamp file
update-files.sh	2020-07-06 15:19	224	

Directory listing dynamically generated by a web server

A web server program may be capable to manage the dynamic generation (on the fly) of a **directory index list** of files and sub-directories.[36]

If a web server program is configured to do so and a requested URL path matches an existing directory and its access is allowed and no static index file is found under that directory then a web page (usually in HTML format), containing the list of files or subdirectories of above mentioned directory, is *dynamically generated* (on the fly). If it cannot be generated an error is returned.

Some web server programs allow the customization of directory listings by allowing the usage of a web page template—an HTML document containing placeholders, (e.g., `$(FILE_NAME)`, `$(FILE_SIZE)`, etc.) that are replaced with the field values of each file entry found in directory by web server (e.g., `index.tpl`) or the usage of HTML and embedded source code that is interpreted and executed (e.g.,, `index.asp`) or by supporting the usage of dynamic index programs such as CGIs, SCGIs, FCGIs (e.g., `index.cgi`, `index.php`, `index.fcgi`).

Usage of dynamically generated *directory listings* is usually avoided or limited to a few selected directories of a website because that generation takes much more OS resources than sending a static index page.

The main usage of *directory listings* is to allow the download of files (usually when their names, sizes, modification date-times or file attributes may change randomly and frequently) *as they are, without requiring to provide further information to requesting user*.[37]

Program or module processing

An external program or an internal module (*processing unit*) can execute some sort of application function that may be used to get data from or to store data to one or more data repositories:

- files (file system);
- databases (DBs);
- other sources located in local computer or in other computers.

A *processing unit* can return any kind of web content, also by using data retrieved from a data repository:

- a document (e.g., HTML, XML, etc.);
- an image;
- a video;
- structured data (e.g., that may be used to update one or more values displayed by a dynamic page (DHTML) of a web interface and that maybe was requested by an XMLHttpRequest API) (see also: dynamic page).

In practice whenever there is content that may vary, depending on one or more parameters contained in client request or in configuration settings, then, usually, it is generated dynamically.

Send response message

Web server programs are able to send response messages as replies to client request messages.[29]

An error response message may be sent because a request message could not be successfully read or decoded or analyzed or executed.[30]

NOTE: the following sections are reported only as examples to help to understand what a web server, more or less, does; these sections are by any means neither exhaustive nor complete.

Error message

A web server program may reply to a client request message with many kinds of error messages, anyway these errors are divided mainly in two categories:

- HTTP client errors, due to the type of request message or to the availability of requested web resource;[38]
- HTTP server errors, due to internal server errors.[39]

When an error response or message is received by a client browser, then if it is related to the main user request (e.g., an URL of a web resource such as a web page) then usually that error message is shown in some browser window or message.

URL authorization

A web server program may be able to verify whether the requested URL path:[40]

- can be freely accessed by everybody;
- requires a user authentication (request of user credentials such as user name and password);
- access is forbidden to some or all kind of users.

If the authorization or access-rights feature has been implemented and enabled and access to web resource is not granted, then, depending on the required access rights, a web server program:

- can deny access by sending a specific error message (e.g., access forbidden);
- may deny access by sending a specific error message (e.g., access unauthorized) that usually forces the client browser to ask human user to provide required user credentials; if authentication credentials are provided then web server program verifies and accepts or rejects them.

URL redirection

A web server program *may* have the capability of doing URL redirections to new URLs (new locations) which consists in replying to a client request message with a response message containing a new URL suited to access a valid or an existing web resource (client should redo the request with the new URL).[41]

URL redirection of location is used:[41]

- to fix a directory name by adding a final slash '/';[36]
- to give a new URL for a no more existing URL path to a new path where that kind of web resource can be found.
- to give a new URL to another domain when current domain has too much load.

Example 1: a URL path points to a **directory** name but it does not have a final slash '/' so web server sends a redirect to client in order to instruct it to redo the request with the fixed path name.[36]

From:

`/directory1/directory2`

To:

`/directory1/directory2/`

Example 2: a whole set of documents has been **moved inside website** in order to reorganize their file system paths.

From:

`/directory1/directory2/2021-10-08/`

To:

`/directory1/directory2/2021/10/08/`

Example 3: a whole set of documents has been **moved to a new website** and now it is mandatory to use secure HTTPS connections to access them.

From:

`http://www.example.com/directory1/directory2/2021-10-08/`

To:

`https://docs.example.com/directory1/2021-10-08/`

Above examples are only a few of the possible kind of redirections.

Successful message

A web server program is able to reply to a valid client request message with a successful message, optionally containing requested **web resource data**.[42]

If web resource data is sent back to client, then it can be **static content** or **dynamic content** depending on how it has been retrieved (from a file or from the output of some program or module).

Content cache

In order to speed up web server responses by lowering average HTTP response times and hardware resources used, many popular web servers implement one or more content caches, each one specialized in a content category.[43] [44]

Content is usually cached by its origin:

- static content:
  - file cache;
- dynamic content:
  - dynamic cache (module or program output).

File cache

Historically, static contents found in files which had to be accessed frequently, randomly and quickly, have been stored mostly on electro-mechanical disks since mid-late 1960s and 1970s; regrettably reads from and writes to those kind of devices have always been considered very slow operations when compared to RAM speed and so, since early OSs, first disk caches and then also OS file cache sub-systems were developed to speed up I/O operations of frequently accessed data.

Even with the aid of an OS file cache, the relative or occasional slowness of I/O operations involving directories and files stored on disks became soon a bottleneck in the increase of performances expected from top level web servers, specially since mid-late 1990s, when web Internet traffic started to grow exponentially along with the constant increase of speed of Internet or network lines.

The problem about how to further efficiently speed-up the serving of static files, thus increasing the maximum number of requests or responses per second (RPS), started to be studied and researched since mid 1990s, with the aim to propose useful cache models that could be implemented in web server programs.[45]

In practice, nowadays, many web server programs include their own *userland* **file cache**, tailored for a web server usage and using their specific implementation and parameters.[46] [47] [48]

The wide spread adoption of RAID and fast solid-state drives (storage hardware with very high I/O speed) has slightly reduced but of course not eliminated the advantage of having a file cache incorporated in a web server.

Dynamic cache

Dynamic content, output by an internal module or an external program, may not always change very frequently (given a unique URL with keys or parameters) and so, maybe for a while (e.g., from one second to several hours or more), the resulting output can be cached in RAM or even on a fast disk.[49]

The typical usage of a dynamic cache is when a website has dynamic web pages about news, weather, images, maps, etc. that do not change frequently (e.g., every *n* minutes) and that are accessed by a huge number of clients per minute per hour; in those cases it is useful to return

cached content too (without calling the internal module or the external program) because clients often do not have an updated copy of the requested content in their browser caches.[50]

Anyway, in most cases those kind of caches are implemented by external servers (e.g., reverse proxy) or by storing dynamic data output in separate computers, managed by specific applications (e.g., memcached), in order to not compete for hardware resources (CPU, RAM, disks) with web servers.[51] [52]

Kernel-mode and user-mode web servers

A web server software can be either incorporated into the OS and executed in kernel space, or it can be executed in user space (like other regular applications).

Web servers that run in kernel mode (usually called kernel space web servers) can have direct access to kernel resources and so they can be, in theory, faster than those running in user mode, but there are disadvantages in running a web server in kernel mode (e.g., difficulties in developing and debugging software) whereas run-time critical errors may lead to serious problems in OS kernel.

Web servers that run in user-mode have to ask the system for permission to use more memory or more CPU resources. Not only do these requests to the kernel take time, but they might not always be satisfied because the system reserves resources for its own usage and has the responsibility to share hardware resources with all the other running applications. Executing in user mode can also mean using more buffer or data copies (between user-space and kernel-space) which can lead to a decrease in the performance of a user-mode web server.

Nowadays almost all web server software is executed in user mode (because many of the aforementioned small disadvantages have been overcome by faster hardware, new OS versions, much faster OS system calls and new optimized web server software). See also comparison of web server software to discover which of them run in kernel mode or in user mode (also referred as kernel space or user space).

Performances

To **improve the user experience** (on client or browser side), a web server should **reply quickly** (as soon as possible) to client requests; unless content response is throttled (by configuration) for some type of files (e.g., big or huge files), also returned data content should be sent as fast as possible (high transfer speed).

In other words, a **web server should always be very responsive**, even under high load of web traffic, in order to keep **total user's wait** (sum of browser time + network time + **web server**

response time) for a response **as low as possible**.

Performance metrics

For web server software, main key **performance metrics** (measured under vary operating conditions) usually are at least the following ones:[53]

- **number of *requests per second*** (**RPS**, similar to **QPS**, depending on HTTP version and configuration, type of HTTP requests and other operating conditions);
- **number of *connections per second*** (**CPS**), is the number of connections per second accepted by web server (useful when using HTTP/1.0 or HTTP/1.1 with a very low limit of requests or responses per connection, i.e., 1 .. 20);
- **network latency** + **response time** for each new client request; usually benchmark tool shows how many requests have been satisfied within a scale of time laps (e.g., within 1ms, 3ms, 5ms, 10ms, 20ms, 30ms, 40ms) or the shortest, the average and the longest response time;
- **throughput of responses**, in bytes per second.

Among the operating conditions, the **number** (1 .. *n*) of **concurrent client connections** used during a test is an important parameter because it allows to correlate the **concurrency level** supported by web server with results of the tested performance metrics.

Software efficiency

The **specific web server software design and model adopted**:

- single process or multi-process;
- single thread (no thread) or multi-thread for each process;
- usage of coroutines or not;

... and other **programming techniques**, such as:

- minimization of possible CPU cache misses;
- minimization of possible CPU branch mispredictions in critical paths for speed;
- minimization of the number of system calls used to perform a certain function or task;
- other tricks;

... used to implement a web server program, **can bias a lot the performances** and in particular the **scalability level** that can be achieved under **heavy load** or when using high end hardware (many CPUs, disks and lots of RAM).

In practice some web server software models may require more OS resources (specially more CPUs and more RAM) than others to be able to work well and so to achieve target performances.

Operating conditions

There are many **operating conditions that can affect the performances** of a web server; performance values may vary depending on:

- the settings of web server (including the fact that log file is or is not enabled, etc.);
- the HTTP version used by client requests;
- the average HTTP request type (method, length of HTTP headers and optional body);
- whether the requested content is static or dynamic;
- whether the content is cached or not cached (by server or client);
- whether the content is compressed on the fly (when transferred), pre-compressed (i.e., when a file resource is stored on disk already compressed so that web server can send that file directly to the network with the only indication that its content is compressed) or not compressed at all;
- whether the connections are or are not encrypted;
- the average network speed between web server and its clients;
- the number of active TCP connections;
- the number of active processes managed by web server (including external CGI, SCGI, FCGI programs);
- the hardware and software limitations or settings of the OS of the computers on which the web server runs;
- other minor conditions.

Benchmarking

Performances of a web server are typically benchmarked by using one or more of the available automated load testing tools.

Load limits

A web server (program installation) usually has pre-defined **load limits** for each combination of operating conditions, also because it is limited by OS resources and because it can handle only a limited number of concurrent client connections (usually between 2 and several tens of

thousands for each active web server process, see also the C10k problem and the C10M problem).

When a web server is near to or over its load limits, it gets **overloaded** and so it may become **unresponsive**.

Causes of overload

At any time web servers can be overloaded due to one or more of the following causes:

- **Excess legitimate web traffic**. Thousands or even millions of clients connecting to the website in a short amount of time (e.g., the Slashdot effect).
- Distributed Denial of Service attacks. A denial-of-service attack (DoS attack) or distributed denial-of-service attack (DDoS attack) is an attempt to make a computer or network resource unavailable to its intended users.
- Computer worms that sometimes cause abnormal traffic because of millions of infected computers (not coordinated among them).
- XSS worms can cause high traffic because of millions of infected browsers or web servers.
- Internet bots Traffic not filtered or limited on large websites with very few network resources (e.g., bandwidth) or hardware resources (CPUs, RAM, disks).
- Internet (network) slowdowns (e.g., due to packet losses) so that client requests are served more slowly and the number of connections increases so much that server limits are reached.
- Web servers, **serving dynamic content**, **waiting for slow responses coming from back-end computers** (e.g., databases), maybe because of too many queries mixed with too many inserts or updates of DB data; in these cases web servers have to wait for back-end data responses before replying to HTTP clients but during these waits too many new client connections or requests arrive and so they become overloaded.
- Web servers (computers) **partial unavailability**. This can happen because of required or urgent maintenance or upgrade, hardware or software failures such as back-end (e.g., database) failures; in these cases the remaining web servers may get too much traffic and become overloaded.

Symptoms of overload

The symptoms of an overloaded web server are usually the following ones:

- Requests are served with (possibly long) delays (from one second to a few hundred seconds).

- The web server returns an HTTP error code, such as 500, 502,[54][55] 503,[56] 504,[57] 408, or even an intermittent 404.
- The web server refuses or resets (interrupts) TCP connections before it returns any content.
- In very rare cases, the web server returns only a part of the requested content. This behavior can be considered a bug, even if it usually arises as a symptom of overload.

Anti-overload techniques

To partially overcome above average load limits and to prevent overload, most popular websites use common techniques like the following ones:

- Tuning OS parameters for hardware capabilities and usage.
- Tuning web servers parameters to improve their security and performances.
- Deploying **web cache** techniques (not only for static contents but, whenever possible, for dynamic contents too).
- Managing network traffic, by using:
  - Firewalls to block unwanted traffic coming from bad IP sources or having bad patterns;
  - HTTP traffic managers to drop, redirect or rewrite requests having bad HTTP patterns;
  - Bandwidth management and traffic shaping, in order to smooth down peaks in network usage.
- Using different domain names, IP addresses and computers to serve different kinds (static and dynamic) of content; the aim is to **separate** big or huge files (`download.*`) (that domain might be replaced also by a CDN) from small and medium-sized files (`static.*`) and from main dynamic site (maybe where some contents are stored in a backend database) (`www.*`); the idea is to be able to efficiently serve big or huge (over 10 – 1000 MB) files (maybe throttling downloads) and to fully cache small and medium-sized files, without affecting performances of dynamic site under heavy load, by using different settings for each (group) of web server computers:
  - `https://download.example.com`
  - `https://static.example.com`
  - `https://www.example.com`
- Using many web servers (computers) that are grouped together behind a load balancer so that they act or are seen as one big web server.
- Adding more hardware resources (i.e., RAM, fast disks) to each computer.
- Using more efficient computer programs for web servers (see also: software efficiency).

- Using the most efficient **Web Server Gateway Interface** to process dynamic requests (spawning one or more external programs every time a dynamic page is retrieved, kills performances).
- Using other programming techniques and workarounds, especially if dynamic content is involved, to speed up the HTTP responses (i.e., by avoiding dynamic calls to retrieve objects, such as style sheets, images and scripts), that never change or change very rarely, by copying that content to static files once and then keeping them synchronized with dynamic content.
- Using latest efficient versions of HTTP (e.g., beyond using common HTTP/1.1 also by enabling HTTP/2 and maybe HTTP/3 too, whenever available web server software has reliable support for the latter two protocols) in order to reduce a lot the number of TCP/IP connections started by each client and the size of data exchanged (because of more compact HTTP headers representation and maybe data compression). This may not prevent overloads of RAM and CPU caused by the need for encryption. It may also not address overloads caused by excessively large files uploaded at high speed, because they are optimized for concurrency.[58][59]

Market share

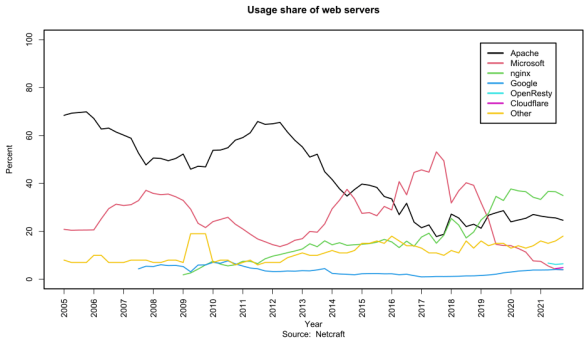


Chart:
Market share of all sites for most popular web servers 2005–2021

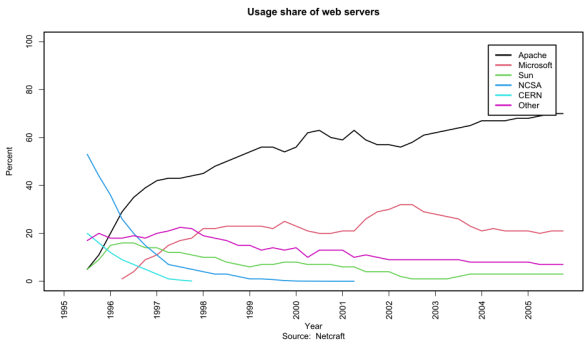


Chart:
Market share of all sites for most popular web servers 1995–2005

Below are the latest statistics of the market share of all sites of the top web servers on the Internet by Netcraft.

Web server: Market share of all sites

Date	nginx (Nginx, Inc.)	Apache (ASF)	OpenResty (OpenResty Software Foundation)	Cloudflare Server (Cloudflare, Inc.)	IIS (Microsoft)	GWS (Google)	Others
October 2021[60]	34.95%	24.63%	6.45%	4.87%	4.00% (*)	4.00% (*)	Less than 22%
February 2021[61]	34.54%	26.32%	6.36%	5.0%	6.5%	3.90%	Less than 18%
February 2020[62]	36.48%	24.5%	4.00%	3.0%	14.21%	3.18%	Less than 15%
February 2019[63]	25.34%	26.16%	N/A	N/A	28.42%	1.66%	Less than 19%
February 2018[64]	24.32%	27.45%	N/A	N/A	34.50%	1.20%	Less than 13%
February 2017[65]	19.42%	20.89%	N/A	N/A	43.16%	1.03%	Less than 15%
February 2016[66]	16.61%	32.80%	N/A	N/A	29.83%	2.21%	Less than 19%

NOTE: (*) percentage rounded to integer number, because its decimal values are not publicly reported by source page (only its rounded value is reported in graph).

See also

- Server (computing)
- Application server
- Comparison of web server software
- HTTP server (core part of a web server program that serves HTTP requests)
- HTTP compression
- Web application
- Open source web application
- List of AMP packages
- Variant object
- Virtual hosting
- Web hosting service

- Web container
- Web proxy
- Web service

Standard Web Server Gateway Interfaces used for dynamic contents:

- CGI Common Gateway Interface
- SCGI Simple Common Gateway Interface
- FastCGI Fast Common Gateway Interface

A few other Web Server Interfaces (server or programming language specific) used for dynamic contents:

- SSI Server Side Includes, rarely used, static HTML documents containing SSI directives are interpreted by server software to include small dynamic data on the fly when pages are served (e.g., date and time, other static file contents, etc.).
- SAPI Server Application Programming Interface:
  - ISAPI Internet Server Application Programming Interface
  - NSAPI Netscape Server Application Programming Interface
- PSGI Perl Web Server Gateway Interface
- WSGI Python Web Server Gateway Interface
- Rack Rack Web Server Gateway Interface
- JSGI JavaScript Web Server Gateway Interface
- Java Servlet, JavaServer Pages
- Active Server Pages, ASP.NET

References

1. Nancy J. Yeager; Robert E. McGrath (1996). *Web Server Technology* (https://books.google.com/books?id=0jExRH3_-hQC&q=%22Web+server%22+-wikipedia&pg=PA14) . Morgan Kaufmann. ISBN 1-55860-376-X. Archived (https://web.archive.org/web/20230120185216/https://books.google.com/books?id=0jExRH3_-hQC&q=%22Web+server%22+-wikipedia&pg=PA14) from the original on 20 January 2023. Retrieved 22 January 2021.

2. William Nelson; Arvind Srinivasan; Murthy Chintalapati (2009). *Sun Web Server: The Essential Guide* (https://books.google.com/books?id=EgNKKPxK9fgC&dq=%22Web+server%22+-wikipedia&pg=PT549) . Pearson Education. ISBN 978-0-13-712892-1. Archived (https://web.archive.org/web/20230120185217/https://www.google.it/books/edition/Sun_Web_Server/EgNKKPxK9fgC?hl=en&gbpv=1&dq=%22Web+server%22+-wikipedia&pg=PT549&printsec=frontcover) from the original on 20 January 2023. Retrieved 14 October 2021.
3. "What is a web server? - MDN Web Docs" (https://developer.mozilla.org/en-US/docs/Learn_web_development/Howto/Web_mechanics/What_is_a_web_server) . *MDN Web Docs*. 13 March 2025. Retrieved 20 March 2025.
4. "What Is a Web Server and How Does It Work? - IT Glossary | SolarWinds" (https://www.solarwinds.com/resources/it-glossary/web-server) . *www.solarwinds.com*. Retrieved 26 April 2025.
5. "What are webservers?" (https://www.akamai.com/glossary/what-are-web-servers) . *Akamai.com*. 26 April 2025.
6. "Study Material on Web Servers - eGyanKosh" (https://egyankosh.ac.in/bitstream/123456789/72095/1/Unit-9.pdf) (PDF). *eGyanKosh*. Retrieved 20 March 2025.
7. "Short History of the Web" (https://home.web.cern.ch/science/computing/birth-web/short-history-web) . *CERN*. Retrieved 20 March 2025.
8. Zolfagharifard, Ellie (24 November 2018). " 'Father of the web' Sir Tim Berners-Lee on his plan to fight fake news" (https://www.telegraph.co.uk/technology/2018/11/24/father-web-sir-tim-berners-lee-plan-fight-fake-news/) . *The Telegraph*. London. ISSN 0307-1235 (https://search.worldcat.org/issn/0307-1235) . Archived (https://ghostarchive.org/archive/20220111/https://www.telegraph.co.uk/technology/2018/11/24/father-web-sir-tim-berners-lee-plan-fight-fake-news/) from the original on 11 January 2022. Retrieved 1 February 2019.
9. "History of Computers and Computing, Internet, Birth, The World Wide Web of Tim Berners-Lee" (http://history-computer.com/Internet/Maturing/Lee.html) . *history-computer.com*. Archived (https://web.archive.org/web/20190104193211/http://history-computer.com/Internet/Maturing/Lee.html) from the original on 4 January 2019. Retrieved 1 February 2019.
10. Tim Berner-Lee (1992). "WWW Project History (original)" (http://info.cern.ch/hypertext/WWW/History.html) . CERN (World Wide Web project). Archived (https://web.archive.org/web/20211208000457/http://info.cern.ch/hypertext/WWW/History.html) from the original on 8 December 2021. Retrieved 20 December 2021.

11. Tim Berner-Lee (20 August 1991). "WorldWideWeb wide-area hypertext app available (announcement)" (https://groups.google.com/g/comp.sys.next.announce/c/avWAjISncfw?pli=1) . CERN (World Wide Web project). Archived (https://web.archive.org/web/20211202204104/https://groups.google.com/g/comp.sys.next.announce/c/avWAjISncfw?pli=1) from the original on 2 December 2021. Retrieved 16 October 2021.
12. Web Administrator. "Web History" (https://web30.web.cern.ch/web-history.html) . CERN (World Wide Web project). Archived (https://web.archive.org/web/20211202001216/https://web30.web.cern.ch/web-history.html) from the original on 2 December 2021. Retrieved 16 October 2021.
13. Tim Berner-Lee (2 August 1991). "Qualifiers on hypertext links ..." (https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt) CERN (World Wide Web project). Archived (https://web.archive.org/web/20211207032603/https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt) from the original on 7 December 2021. Retrieved 16 October 2021.
14. Tim Smith; François Flückiger. "Licensing the Web" (https://home.cern/science/computing/birth-web/licensing-web) . CERN (World Wide Web project). Archived (https://web.archive.org/web/20211206161306/http://home.cern/science/computing/birth-web/licensing-web) from the original on 6 December 2021. Retrieved 16 October 2021.
15. Ali Mesbah (2009). *Analysis and Testing of Ajax-based Single-page Web Applications* (https://www.researchgate.net/publication/27353046) . ISBN 978-90-79982-02-8. Retrieved 18 December 2021.
16. Robert H'obbes' Zakon. "Hobbes' Internet Timeline v5.1 (WWW Growth) NOTE: till 1996 number of web servers = number of web sites" (https://web.archive.org/web/20000815100731/http://www.isoc.org/guest/zakon/Internet/History/HIT.html) . ISOC. Archived from the original on 15 August 2000. Retrieved 18 December 2021.
17. "NCSA httpd" (https://web.archive.org/web/20100801142847/http://illinois.edu/lb/imageList/2943) . NCSA (web archive). Archived from the original (http://illinois.edu/lb/imageList/2943) on 1 August 2010. Retrieved 16 December 2021.
18. "About the Apache HTTPd server: How Apache Came to be" (https://httpd.apache.org/ABOUT_APACHE.html) . Apache: HTTPd server project. 1997. Archived (https://web.archive.org/web/20080607122013/http://httpd.apache.org/ABOUT_APACHE.html) from the original on 7 June 2008. Retrieved 17 December 2021.
19. "Web Server Survey, NOTE: number of active web sites in year 2000 has been interpolated" (https://news.netcraft.com/archives/2021/12/22/december-2021-web-server-survey.html) . Netcraft. 22 December 2021. Archived (https://web.archive.org/web/20211227070918/https://news.netcraft.com/archives/2021/12/22/december-2021-web-server-survey.html) from the original on 27 December 2021. Retrieved 27 December 2021.

20. "Netcraft: web server software (1996)" (https://web.archive.org/web/19961230090855/http://www.netcraft.com/survey/servers.html) . Netcraft (web archive). Archived from the original (http://www.netcraft.com/survey/servers.html) on 30 December 1996. Retrieved 16 December 2021.

21. "Overview of new features in Apache 2.2" (https://httpd.apache.org/docs/2.2/new_features_2_2.html) . Apache: HTTPd server project. 2005. Archived (https://web.archive.org/web/20211127091204/http://httpd.apache.org/docs/2.2/new_features_2_2.html) from the original on 27 November 2021. Retrieved 16 December 2021.

22. "Overview of new features in Apache 2.4" (https://httpd.apache.org/docs/2.4/new_features_2_4.html) . Apache: HTTPd server project. 2012. Archived (https://web.archive.org/web/20211126112829/http://httpd.apache.org/docs/2.4/new_features_2_4.html) from the original on 26 November 2021. Retrieved 16 December 2021.

23. "Connections, persistent connections: practical considerations" (https://datatracker.ietf.org/doc/html/rfc2616#section-8.1.4) . *RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1* (https://datatracker.ietf.org/doc/html/rfc2616) . pp. 46–47. sec. 8.1.4. doi:10.17487/RFC2616 (https://doi.org/10.17487%2FRFC2616) . RFC 2616 (https://datatracker.ietf.org/doc/html/rfc2616) .

24. "Maximum concurrent connections to the same domain for browsers" (http://sgdev-blog.blogspot.com/2014/01/maximum-concurrent-connection-to-same.html) . 2017. Archived (https://web.archive.org/web/20211221234815/http://sgdev-blog.blogspot.com/2014/01/maximum-concurrent-connection-to-same.html) from the original on 21 December 2021. Retrieved 21 December 2021.

25. "Linux Web Server Performance Benchmark - 2016 results" (https://www.rootusers.com/linux-web-server-performance-benchmark-2016-results/) . RootUsers. 8 March 2016. Archived (https://web.archive.org/web/20211223131547/https://www.rootusers.com/linux-web-server-performance-benchmark-2016-results/) from the original on 23 December 2021. Retrieved 22 December 2021.

26. "Will HTTP/2 replace HTTP/1.x?" (https://http2.github.io/faq/#will-http2-replace-http1x) . IETF HTTP Working Group. Archived (https://web.archive.org/web/20140927004541/http://http2.github.io/faq/#will-http2-replace-http1x) from the original on 27 September 2014. Retrieved 22 December 2021.

27. "Implementations of HTTP/2 in client and server software" (https://github.com/httpwg/http2-spec/wiki/Implementations) . IETF HTTP Working Group. Archived (https://web.archive.org/web/20211223003728/https://github.com/httpwg/http2-spec/wiki/Implementations) from the original on 23 December 2021. Retrieved 22 December 2021.

28. "Why just one TCP connection?" (https://http2.github.io/faq/#why-just-one-tcp-connection) . IETF HTTP Working Group. Archived (https://web.archive.org/web/20140927004541/http://http2.github.io/faq/#why-just-one-tcp-connection) from the original on 27 September 2014. Retrieved 22 December 2021.

29. "Client/Server Messaging" (https://datatracker.ietf.org/doc/html/rfc7230#section-2.1) . *RFC 7230, HTTP/1.1: Message Syntax and Routing* (https://datatracker.ietf.org/doc/html/rfc7230) . pp. 7–8. sec. 2.1. doi:10.17487/RFC7230 (https://doi.org/10.17487%2FRFC7230) . RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) .

30. "Handling Incomplete Messages" (https://datatracker.ietf.org/doc/html/rfc7230#section-3.4) . *RFC 7230, HTTP/1.1: Message Syntax and Routing* (https://datatracker.ietf.org/doc/html/rfc7230) . p. 34. sec. 3.4. doi:10.17487/RFC7230 (https://doi.org/10.17487%2FRFC7230) . RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) .

31. "Message Parsing Robustness" (https://datatracker.ietf.org/doc/html/rfc7230#section-3.5) . *RFC 7230, HTTP/1.1: Message Syntax and Routing* (https://datatracker.ietf.org/doc/html/rfc7230) . pp. 34–35. sec. 3.5. doi:10.17487/RFC7230 (https://doi.org/10.17487%2FRFC7230) . RFC 7230 (https://datatracker.ietf.org/doc/html/rfc7230) .

32. R. Bowen (29 September 2002). "URL Mapping" (https://people.apache.org/~jim/ApacheCons/ApacheCon2002/pdf/Bowen-urlmap-ACUS02/bowen-urlmap-ACUS02.pdf) (PDF). Apache software foundation. Archived (https://web.archive.org/web/20211115181448/http://people.apache.org/~jim/ApacheCons/ApacheCon2002/pdf/Bowen-urlmap-ACUS02/bowen-urlmap-ACUS02.pdf) (PDF) from the original on 15 November 2021. Retrieved 15 November 2021.

33. "Mapping URLs to Filesystem Locations" (https://httpd.apache.org/docs/2.4/urlmapping.html) . Apache: HTTPd server project. 2021. Archived (https://web.archive.org/web/20211020053640/http://httpd.apache.org/docs/2.4/urlmapping.html) from the original on 20 October 2021. Retrieved 19 October 2021.

34. "Dynamic Content with CGI" (https://httpd.apache.org/docs/2.4/howto/cgi.html) . Apache: HTTPd server project. 2021. Archived (https://web.archive.org/web/20211115181448/https://httpd.apache.org/docs/2.4/howto/cgi.html) from the original on 15 November 2021. Retrieved 19 October 2021.

35. Chris Shiflett (2003). *HTTP developer's handbook* (https://books.google.com/books?id=oxg8_i9dVakC&pg=PA38) . Sams's publishing. ISBN 0-672-32454-7. Archived (https://web.archive.org/web/20230120185219/https://www.google.it/books/edition/HTTP_Developer_s_Handbook/oxg8_i9dVakC?hl=en&gbpv=1&pg=PA38&printsec=frontcover) from the original on 20 January 2023. Retrieved 9 December 2021.

36. ASF Infrabot (22 May 2019). "Directory listings" (https://cwiki.apache.org/confluence/display/HTTPD/DirectoryListings) . Apache foundation: HTTPd server project. Archived (https://web.archive.org/web/20190607234544/https://cwiki.apache.org/confluence/display/HTTPD/DirectoryListings) from the original on 7 June 2019. Retrieved 16 November 2021.

37. "Apache: directory listing to download files" (https://archive.apache.org/dist/httpd/) . Apache: HTTPd server. Archived (https://web.archive.org/web/20211202004258/http://archive.apache.org/dist/httpd/) from the original on 2 December 2021. Retrieved 16 December 2021.

38. "Client Error 4xx" (https://datatracker.ietf.org/doc/html/rfc7231#section-6.5) . *RFC 7231, HTTP/1.1: Semantics and Content* (https://datatracker.ietf.org/doc/html/rfc7231) . p. 58. sec. 6.5. doi:10.17487/RFC7231 (https://doi.org/10.17487%2FRFC7231) . RFC 7231 (https://datatracker.ietf.org/doc/html/rfc7231) .

39. "Server Error 5xx" (https://datatracker.ietf.org/doc/html/rfc7231#section-6.6) . *RFC 7231, HTTP/1.1: Semantics and Content* (https://datatracker.ietf.org/doc/html/rfc7231) . pp. 62-63. sec. 6.6. doi:10.17487/RFC7231 (https://doi.org/10.17487%2FRFC7231) . RFC 7231 (https://datatracker.ietf.org/doc/html/rfc7231) .

40. "Introduction" (https://datatracker.ietf.org/doc/html/rfc7235#section-1) . *RFC 7235, HTTP/1.1: Authentication* (https://datatracker.ietf.org/doc/html/rfc7235) . p. 3. sec. 1. doi:10.17487/RFC7235 (https://doi.org/10.17487%2FRFC7235) . RFC 7235 (https://datatracker.ietf.org/doc/html/rfc7235) .

41. "Response Status Codes: Redirection 3xx" (https://datatracker.ietf.org/doc/html/rfc7231#section-6.4) . *RFC 7231, HTTP/1.1: Semantics and Content* (https://datatracker.ietf.org/doc/html/rfc7231) . pp. 53–54. sec. 6.4. doi:10.17487/RFC7231 (https://doi.org/10.17487%2FRFC7231) . RFC 7231 (https://datatracker.ietf.org/doc/html/rfc7231) .

42. "Successful 2xx" (https://datatracker.ietf.org/doc/html/rfc7231#section-6.3) . *RFC 7231, HTTP/1.1: Semantics and Content* (https://datatracker.ietf.org/doc/html/rfc7231) . pp. 51-54. sec. 6.3. doi:10.17487/RFC7231 (https://doi.org/10.17487%2FRFC7231) . RFC 7231 (https://datatracker.ietf.org/doc/html/rfc7231) .

43. "Caching Guide" (https://httpd.apache.org/docs/2.4/caching.html) . Apache: HTTPd server project. 2021. Archived (https://web.archive.org/web/20211209211243/https://httpd.apache.org/docs/2.4/caching.html) from the original on 9 December 2021. Retrieved 9 December 2021.

44. "NGINX Content Caching" (https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/) . F5 NGINX. 2021. Archived (https://web.archive.org/web/20211209211246/https://docs.nginx.com/nginx/admin-guide/content-cache/content-caching/) from the original on 9 December 2021. Retrieved 9 December 2021.

45. Evangelos P. Markatos (1996). "Main Memory Caching of Web Documents" (https://www.ra.ethz.ch/cdstore/www5/www218/overview.htm) . Computer networks and ISDN Systems. Archived (https://web.archive.org/web/20230120185224/https://www.ra.ethz.ch/cdstore/www5/www218/overview.htm) from the original on 20 January 2023. Retrieved 9 December 2021.

46. "IPlanet Web Server 7.0.9: file-cache" (https://docs.oracle.com/cd/E19146-01/821-1827/gaidp/index.html) . Oracle. 2010. Archived (https://web.archive.org/web/20211209175035/https://docs.oracle.com/cd/E19146-01/821-1827/gaidp/index.html) from the original on 9 December 2021. Retrieved 9 December 2021.

47. "Apache Module mod_file_cache" (https://httpd.apache.org/docs/2.4/mod/mod_file_cache.html) . Apache: HTTPd server project. 2021. Archived (https://web.archive.org/web/20211209194811/https://httpd.apache.org/docs/2.4/mod/mod_file_cache.html) from the original on 9 December 2021. Retrieved 9 December 2021.

48. "HTTP server: configuration: file cache" (https://www.gnu.org/software/serveez/manual/html_node/HTTP-Server.html) . GNU. 2021. Archived (https://web.archive.org/web/20211209173634/https://www.gnu.org/software/serveez/manual/html_node/HTTP-Server.html) from the original on 9 December 2021. Retrieved 9 December 2021.

49. "Apache Module mod_cache_disk" (https://httpd.apache.org/docs/2.4/mod/mod_cache_disk.html) . Apache: HTTPd server project. 2021. Archived (https://web.archive.org/web/20211209211241/https://httpd.apache.org/docs/2.4/mod/mod_cache_disk.html) from the original on 9 December 2021. Retrieved 9 December 2021.

50. "What is dynamic cache?" (https://www.educative.io/edpresso/what-is-dynamic-cache) . Educative. 2021. Archived (https://web.archive.org/web/20211209234355/https://www.educative.io/edpresso/what-is-dynamic-cache) from the original on 9 December 2021. Retrieved 9 December 2021.

51. "Dynamic Cache Option Tutorial" (https://www.siteground.com/tutorials/supercacher/dynamic-cache/) . Siteground. 2021. Archived (https://web.archive.org/web/20230120185251/https://www.siteground.com/tutorials/supercacher/dynamic-cache/) from the original on 20 January 2023. Retrieved 9 December 2021.

52. Arun Iyengar; Jim Challenger (2000). "Improving Web Server Performance by Caching Dynamic Data" (https://www.researchgate.net/publication/2585583) . Usenix. Retrieved 9 December 2021.

53. Jussara M. Almeida; Virgilio Almeida; David J. Yates (7 July 1997). "WebMonitor: a tool for measuring World Wide Web server performance" (https://firstmonday.org/ojs/index.php/fm/article/view/539/460) . *First Monday*. doi:10.5210/fm.v2i7.539 (https://doi.org/10.5210%2Ffm.v2i7.539) . Archived (https://web.archive.org/web/20211104215116/https://firstmonday.org/ojs/index.php/fm/article/view/539/460) from the original on 4 November 2021. Retrieved 4 November 2021.

54. Fisher, Tim; Lifewire. "Getting a 502 Bad Gateway Error? Here's What to Do" (https://www.lifewire.com/502-bad-gateway-error-explained-2622939) . *Lifewire*. Archived (https://web.archive.org/web/20170223042443/https://www.lifewire.com/502-bad-gateway-error-explained-2622939) from the original on 23 February 2017. Retrieved 1 February 2019.

55. "What is a 502 bad gateway and how do you fix it?" (https://www.itpro.co.uk/go/30258) . *IT PRO*. Archived (https://web.archive.org/web/20230120185257/https://www.itpro.co.uk/web-hosting/30258/what-is-a-502-bad-gateway-and-how-do-you-fix-it) from the original on 20 January 2023. Retrieved 1 February 2019.

56. Fisher, Tim; Lifewire. "Getting a 503 Service Unavailable Error? Here's What to Do" (https://www.lifewire.com/503-service-unavailable-explained-2622940) . *Lifewire*. Archived (https://web.archive.org/web/20230120185318/https://www.lifewire.com/503-service-unavailable-explained-2622940) from the original on 20 January 2023. Retrieved 1 February 2019.

57. Fisher, Tim; Lifewire. "Getting a 504 Gateway Timeout Error? Here's What to Do" (https://www.lifewire.com/504-gateway-timeout-error-explained-2622941) . *Lifewire*. Archived (https://web.archive.org/web/20210423182953/https://www.lifewire.com/504-gateway-timeout-error-explained-2622941) from the original on 23 April 2021. Retrieved 1 February 2019.

58. many (24 January 2021). "Slow uploads with HTTP/2" (https://github.com/nextcloud/server/issues/25297) . github. Archived (https://web.archive.org/web/20211116002101/https://github.com/nextcloud/server/issues/25297) from the original on 16 November 2021. Retrieved 15 November 2021.

59. Junho Choi (24 August 2020). "Delivering HTTP/2 upload speed improvements" (https://blog.cloudflare.com/delivering-http-2-upload-speed-improvements/) . Cloudflare. Archived (https://web.archive.org/web/20211116002101/https://blog.cloudflare.com/delivering-http-2-upload-speed-improvements/) from the original on 16 November 2021. Retrieved 15 November 2021.

60. "October 2021 Web Server Survey" (https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html) . Netcraft. 15 October 2021. Archived (https://web.archive.org/web/20211115125905/https://news.netcraft.com/archives/2021/10/15/october-2021-web-server-survey.html) from the original on 15 November 2021. Retrieved 15 November 2021.

61. "February 2021 Web Server Survey" (https://news.netcraft.com/archives/2021/02/26/february-2021-web-server-survey.html) . Netcraft. 26 February 2021. Archived (https://web.archive.org/web/20210415082427/https://news.netcraft.com/archives/2021/02/26/february-2021-web-server-survey.html) from the original on 15 April 2021. Retrieved 8 April 2021.
62. "February 2020 Web Server Survey" (https://news.netcraft.com/archives/2020/02/20/february-2020-web-server-survey.html) . Netcraft. 20 February 2020. Archived (https://web.archive.org/web/20210417141149/https://news.netcraft.com/archives/2020/02/20/february-2020-web-server-survey.html) from the original on 17 April 2021. Retrieved 8 April 2021.
63. "February 2019 Web Server Survey" (https://news.netcraft.com/archives/2019/02/28/february-2019-web-server-survey.html) . Netcraft. 28 February 2019. Archived (https://web.archive.org/web/20210415083053/https://news.netcraft.com/archives/2019/02/28/february-2019-web-server-survey.html) from the original on 15 April 2021. Retrieved 8 April 2021.
64. "February 2018 Web Server Survey" (https://news.netcraft.com/archives/2018/02/13/february-2018-web-server-survey.html) . Netcraft. 13 February 2018. Archived (https://web.archive.org/web/20210417141318/https://news.netcraft.com/archives/2018/02/13/february-2018-web-server-survey.html) from the original on 17 April 2021. Retrieved 8 April 2021.
65. "February 2017 Web Server Survey" (https://news.netcraft.com/archives/2017/02/27/february-2017-web-server-survey.html) . Netcraft. 27 February 2017. Archived (https://web.archive.org/web/20170314062733/https://news.netcraft.com/archives/2017/02/27/february-2017-web-server-survey.html) from the original on 14 March 2017. Retrieved 13 March 2017.
66. "February 2016 Web Server Survey" (https://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html) . Netcraft. 22 February 2016. Archived (https://web.archive.org/web/20220127205027/https://news.netcraft.com/archives/2016/02/22/february-2016-web-server-survey.html) from the original on 27 January 2022. Retrieved 27 January 2022.

External links

- Mozilla: what is a web server? (https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server)
- Netcraft: news about web server survey (https://news.netcraft.com/archives/category/web-server-survey/)